Citation

Computer graphics and interactive techniques in Austalasia and South East Asia >archive

Proceedings of the 1st international conference on Computer graphics and interactive techniques in Austalasia and South East Asia >toc

2003 , Melbourne, Australia

SESSION: Sketch >toc

A feature-based pencil drawing method

Authors
Nan Li National University of Singapore, Singapore
Zhiong Huang National University of Singapore, Singapore

Sponsor
SIGGRAPH : ACM Special Interest Group on Computer Graphics and Interactive Techniques

Publisher
ACM Press New York, NY, USA

Pages: 135 - ff Series-Proceeding-Section-Article
Year of Publication: 2003
ISBN:1-58113-578-5

doi> http://doi.acm.org/10.1145/604471.604498 (Use this link to Bookmark this page)

> full text > abstract > references > index terms

> Discuss > Similar > Review this Article Save to Binder

> BibTex Format

FULL TEXT: Access Rules

pdf 1.62 MB

↑ ABSTRACT

We present a method for generating a pencil drawing from a digital image using the feature geometric attributes obtained by analysis of image moment and texture of each region. Thus, the drawing results represent not only the local region information but also the feature characteristics. It is inspired and devised to correspond to a real pencil drawing process: A painter often divides a scene into regions, observes the distinctive features, and applies the pencil strokes accordingly. We have implemented the method and compared the results with Adobe Photoshop 5.0.

↑ REFERENCES

Note: OCR errors may be found in this Reference List extracted from the full text article. ACM has opted to expose the complete List rather than only correct and linked references.

1 B. Cabral and L. C. Leedom, Imaging vector fields using line integral convolution, SIGGRAPH '93, 263-270.

2 F. Durand, V. Ostromoukhov, M. Miller, F. Duranleau, and J. Dorsey, Decoupling strokes and high-level attributes for interactive traditional drawing, Proceedings of the 12th Eurographics Workshop on Rendering, June 2001, 71-82.

3 P. Haeberli, Paint by numbers: Abstract image representations, Computer Graphics, August 1990. 207-214.

4 A. Hertzmann, Painterly rendering with curved brush strokes of multiple sizes, SIGGRAPH '98, 453-480.

5 X. Mao, Y. Nagasaka and A. Imamiya, Automatic generation of pencil drawing from 2D images using line integral convolution, CAD/Graphics 2001, 240-248.

6 R. Mukundan and K. R. Ramakrishnan, Moment functions in image analysis, theory and applications, published by World Scientific in September 1998. ISBN: 9810235240.

7 M. P. Salisbury, S. E. Anderson and D. H. Salesin, Interactive pen-and-ink illustration, SIGGRAPH '94, 101-108.

8 H. W. Shen and D. L. Kao, New line integral convolution algorithm for visualizing time-varying flow fields, IEEE TVCG 4(2), 1998, 98-108.

9 M. Shiraishi and Y. Yamaguchi, An algorithm for automatic painterly rendering based on local source image approximation, Non-Photorealistic Animation and Rendering 2000 (NPAR '00), 53-58.

10 J. A. Shufelt, Texture analysis for enhanced color image quantization, Graphical Models and Image Processing, 59(3), May 1997, 149-163.

11 M. C. Sousa and J. W. Buchanan, Observational model of blenders and erasers in computer-generated pencil rendering, Graphics Interface '99, 157-166.

12 M. C. Sousa and J. W. Buchanan, Computer-generated graphite pencil rendering of 3D polygonal models, EG '99, 195-207.

13 A. H. Vermeulen and P. P. Tanner, PencilSketch-A pencil-based paint system, Graphics Interface '89, 138-143.

14 G. Winkenbach and D. H. Salesin, Computer-generated pen-and ink illustration, SIGGRAPH '94, 91-100.

INDEX TERMS

Primary Classification:
I. Computing Methodologies
- I.3 COMPUTER GRAPHICS
  - I.3.3 Picture/Image Generation
    - Subjects: Display algorithms

Additional Classification:
I. Computing Methodologies
- I.4 IMAGE PROCESSING AND COMPUTER VISION
  - I.4.6 Segmentation
    - Subjects: Pixel classification

Keywords:
enhanced realistic, image moment function, non-photo realistic rendering, texture analysis

---

Artistic Vision: Painterly Rendering Using Computer Vision Techniques

Bruce Gooch
University of Utah

Greg Coombe
University of North Carolina at Chapel Hill

Peter Shirley
University of Utah

Abstract

We present a method that takes a raster image as input and produces a painting-like image composed of strokes rather than pixels. Our method works by first segmenting the image into features, finding the approximate medial axes of these features, and using the medial axes to guide brush stroke creation. System parameters may be interactively manipulated by a user to effect image segmentation, brush stroke characteristics, stroke size, and stroke frequency. This process creates images reminiscent of those contemporary representational painters whose work has an abstract or sketchy quality. Our software is available at http://www.cs.utah.edu/npr/ArtisticVision.

CR Categories: I.3.7 [Computing Methodologies]: Computer Graphics—2D Graphics

Keywords: image moments, image processing, medial axis, non-photorealistic rendering, painting

1 Introduction

The art of painting relies on representation and abstraction. In representational painting, the abstraction occurs when the detail of real images is approximated with limited spatial resolution (brush strokes) and limited chromatic resolution (palette). Economy is a quality of many paintings, and refers to the use of only those brush strokes and colors needed to convey the essence of a scene. This notion of economy has been elusive for computer-painting algorithms. We explore an automated painting algorithm that attempts to achieve economy, particularly in its use of brush strokes.

There are two tasks involved in the creation of a digital painting. First is the creation of brush stroke positions, the second is the rendering of brush strokes. If the brush stroke positions are manually created by a user, then this is a classic paint program. If the brush stroke positions are computed algorithmically, then this is an automatic painting system. In either case, once the brush stroke geometry is known, the brush strokes must then be rendered, usually simulating the physical nature of paint and canvas [4, 17, 24].

We review digital painting strategies in Section 2 and give an overview of our algorithm in Section 3. The conversion from a single segment of an image to a set of planned brush strokes, which is the core of our contribution, is covered in Sections 4–8. We discuss possible extensions to our method in Section 9.

2 Background

Two basic approaches to digital painting are used in computer graphics. The first simulates the characteristics of an artistic medium such as paint on canvas. The second attempts to automatically create paintings by simulating the artistic process. These approaches can be combined because they deal with different aspects, one low-level and one high-level, of painting.

Work intended to simulate artistic media can be further divided into those which simulate the physics a particular medium, and those which just simulate the look. Strassmann [24] simulated the

Figure 1: *A landscape painting of Hovenweep National Monument. This painting was automatically created using the system described in this paper. The input was a scanned vacation photograph.*

look of traditional sumi-e painting with polylines and a raster algorithm. Pham [17] augmented this algorithm using B-splines and offset curves instead of polylines to achieve smoother brush paths. Williams [27] provides a method of merging painting and sculpting by using the raster image as a height field.

Smith [22] points out that by using a scale-invariant primitive for a brush stroke, multi-resolution paintings can be made. Berman et al. [1] showed that multi-resolution painting methods are efficient in both speed and storage. Perlin and Velho [16] used multi-resolution procedural textures to create realistic detail at any scale.

Several authors have simulated the interaction of paper/canvas and a drawing/painting instrument. Cockshott [4] simulated the substrate, diffusion, and gravity in a physically-based paint system. Curtis et al. [6] modeled fluid flow, absorption, and particle distribution to simulate watercolor. Sousa and Buchanan [23] simulated pencil drawing by modeling the physics and interaction of pencil lead, paper, and blending tools.

While the works discussed above are concerned with the low-level interaction of pigment with paper or canvas, other authors aid a user in the creation of an art work, or automate the artistic process. Haeberli [8] built a paint system that re-samples a digital image based on a brush. Wong [28] built a system for charcoal drawing that prompts the user for input at critical stages of the artistic process. Gooch et al. [7] automatically generated technical illustrations from polygonal models of CAD parts. Meier [15] produced painterly animations using a particle system. Litwinowicz [14] produced impressionist-style video by re-sampling a digital

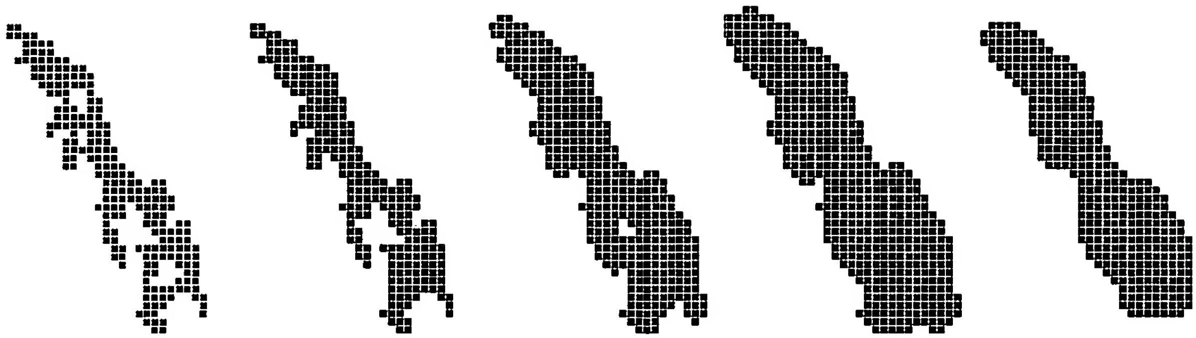


Figure 2: *From left to right: An example of a segmented region. The region after the hole filling algorithm has been run. The region after being opened. The region after being opened again. The region after having been closed.*

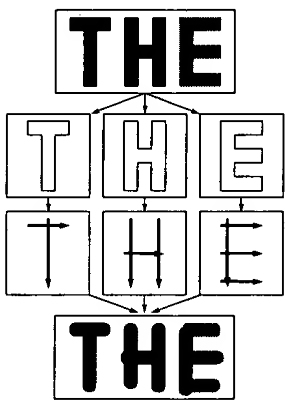


Figure 3: *First an image is segmented using flood filling. Next each segment is independently decomposed into brush stroke paths. Finally brush strokes are rendered to a raster image.*

image and tracking optical flow. Hertzmann [11] refined Haeberli's technique by using progressively smaller brushes to automatically create a hand-painted effect. Shiraishi et al. [21] use image moments to create digital paintings automatically.

The algorithm described in this paper should be grouped with the latter set of works that simulate the high-level results of the artistic process. Our technique results in a resolution-independent list of brush strokes which can be rendered into raster images using any brush stroke rendering method.

3 Algorithm

Our system takes as input a digital image which is first segmented using flood filling (Sec 4). These segments are used to compute brush stroke paths. Using image processing algorithms, the boundaries of each segmented region are smoothed and any holes in the region are filled. The system next finds a discrete approximation to the central axis of each segment called the ridge set (Sec 5). Elements of the ridge set are pieced together into tokens (Sec 6). These tokens can, at the user's discretion, be spatially sorted into ordered lists. In the final image this second sorting has the effect of painting a region with a single large stroke instead of many small strokes. Finally the brush paths are rendered as brush strokes (Sec 7). Figure 3 shows a diagram of our algorithm.

4 Segmentation and Smoothing

A technique sometimes used by artists is to first produce tonal sketches of the scene being painted [22]. These sketches are in effect low resolution gray scale images of the final painting that are used as a reference for form and lighting. Guided by this observation, we employ an intensity-based method of segmenting images.

An image is segmented by selecting an unvisited pixel, and searching outward for nearby unvisited pixels with similar intensities. "Similar intensity" is defined by the number of segmentation levels, and is based on a perceptual metric [9]. Pixels are marked as visited, and a new unvisited pixel is chosen. This is repeated until the entire image is quantized into independent segments.

Segments of similar intensity (such as the letters *T* and *H* in Figure 3) are not connected. This was useful in highly complex regions (such as the forest in Figure 4) because we were able to use the higher-level merging operations to chose stroke size rather than the intensity operations. We experimented with a number of segmentation operations, and found that this simple one was the most flexible. However, given the importantance of segmentation on the final result, we think that more sophisticated segmentation strategies might prove beneficial.

4.1 Hole Filling

The segments produced in the previous step often have small holes due to shadows or highlights in the image. We use a standard hole-filling technique to remove these. Segmented regions are stored as boolean arrays with *true* indicating membership in the region. Each *false* pixel is queried to find how many *true* neighbors it has. Pixels with more than five *true* neighbors are changed to *true*. If a pixel has less than five neighbors, but is enclosed by an all *true* region, then the pixel is also set to *true*.

4.2 Morphological Operations

Opening and closing operations are used to smooth the boundary and remove noise. The Open operation changes every pixel from *false* to *true* if any of its neighbors are *true*. The Close operation changes every pixel from *true* to *false* if any of its neighbors are *false*. In practice we have found that two Open operations followed by a Close operation produced good results. The entire smooting sequence is illustrated in Figure 2. Open and Close operations may also be applied to the approximate medial axis (described in the next section) with good results.

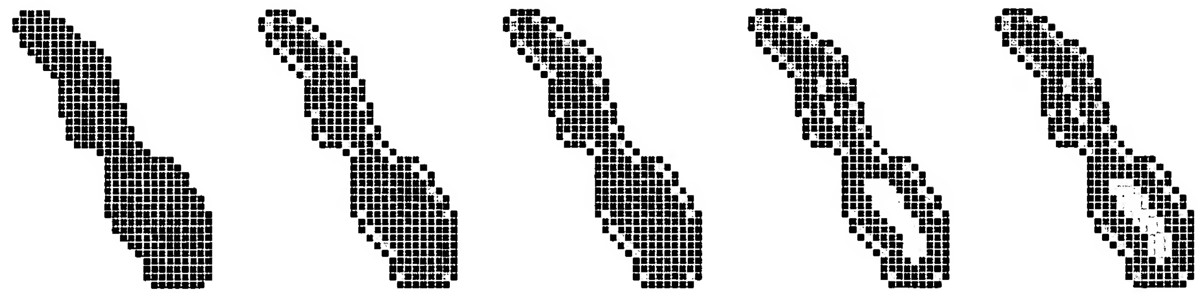


Figure 5: *An example of a distance transform on a region from left to right. First each pixel in the region is initialized to 1 if it is in the segmented region, and to 0 if it is not. Next multiple passes are made over the region. At each pass, a pixel value is incremented if all non-zero neighbors have values less than or equal to the pixel's current value. This example shows the increasing value of the pixels by changing the pixel color to a lighter gray.*

Figure 4: *An example of varying the number of gray levels in image segmentation. Top: the segmented image using 72 gray levels. Bottom: the segmented image using 48 gray levels. Note that the clouds have faded into the sky in the 48 gray level image.*

5 Ridge Set Extraction

The medial axis of each segmented region is computed and used as a basis for the brush stroke rendering algorithm. The medial axis of a region is essentially the "skeleton" of an object. The medial axis transform yields scale and rotation invariant measures for a segment, which is a big advantage over other systems which rely upon grids. The medial axis was first presented by Blum [2], and has been shown to be useful in approximating 2D [13] and 3D objects [12]. The medial axis has also been shown to be a good approximation of the way the human visual system perceives shape [3].

While the medial axis is a continuous representation, there are several types of algorithms for computing the medial axis in image space, including thinning algorithms and distance transforms. Distance transform algorithms are not as sensitive to boundary noise and produce width information, but tend to produce double lines and don't preserve connectedness [13]. Thinning algorithms, such as Rosenfeld's parallel algorithm [18], preserve the connectedness and produce smooth medial axis lines. However these algorithms are sensitive to boundary noise, which will result in undesirable spurs along the medial axis. Another drawback to thinning algorithms is that they do not produce information about the distance to the boundary, which we need to estimate brush stroke width.

The positive aspects of both techniques are combined in our system to form a hybrid method. We first apply the distance transform to extract a discrete approximation of the medial axis called the ridge set. We then thin the ridge set to remove spurs (caused by boundary noise) and double lines (caused when the medial axis falls between pixels). This combination of techniques results in a ridge set with distance information, and reduced sensitivity to noise along the boundary.

5.1 Distance Transform

For each pixel in the region, we compute the shortest distance to the boundary [13]. On the first pass the distance transform algorithm assigns a value of one to each pixel in the region. Subsequent passes approximate a Euclidean distance transform by finding the smallest non-zero neighbor of the current pixel. If this pixel value is larger than the value of current pixel, it is incremented by 1 ($\sqrt{2}$ if it is a diagonal neighbor). The algorithm proceeds until no values are changed during a pass. This process is shown in Figure 5. The number of passes is proportional to the radius of the largest circle that touches both boundaries.

5.2 Approximate Medial Axis Extraction

The next step is to extract the medial axis from the distance-transformed region. A discrete approximation to the medial axis is the ridge set, which is the set of points that are further away from the boundary of the region than any surrounding points. A pixel is part of the ridge set if its values is greater than or equal to the value of all 8 surrounding pixels [13].

This conservative definition of the ridge set may not yield a connected set of lines, which necessitates some of the later grouping algorithms. However, we found that using a less conservative test resulted in noisy line segments and nervous, uncontrolled brush strokes. This approximation avoids the problems of spurs associated with distance transforms, but produces numerous double lines.

To avoid these discretization problems, we are currently exploring a Voronoi diagram method which produces a continuous medial axis. However, the amount of computation required is usually larger than for the discrete method.

5.3 Thinning the Approximate Medial Axis

To address the problem of double lines in the distance transform, we use Rosenfeld's parallel thinning algorithm [18]. This algorithm removes "redundant" pixels from a binary image by testing small neighborhoods of pixels. Each 8-pixel neighborhood is tested for redundant pixels. We developed a fast, unique test for these pixel neighborhoods; the details are discussed in the Appendix.

The thinning algorithm eliminates double-lines and other noise from the ridge set. The algorithm typically requires two to three passes over the ridge set. Another advantage of this thinning algorithm is that points in the ridge set are guaranteed to have at most three neighbors.

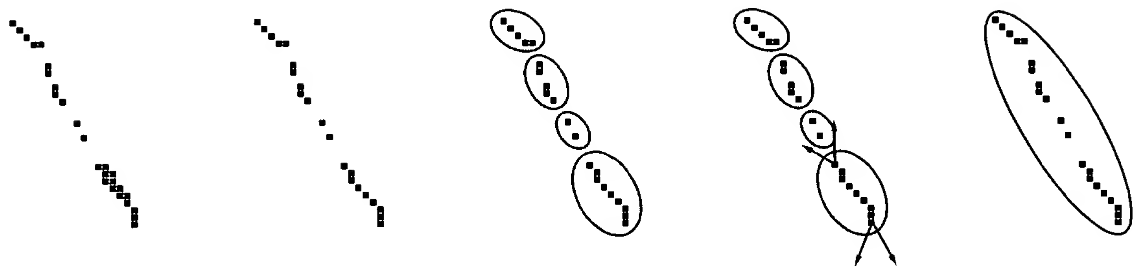


Figure 6: *An example of ridge set extraction, thinning and grouping from left to right. First the ridge set is extracted from the distance transformed image. Second our thinning algorithm is applied. Third the resulting ridge set is grouped into tokens. Forth the tokens are merged into a stroke. The fifth image shows a set of tokens ready to be rendered*

6 Ridge Set Tokenizing and Grouping

The combination of the distance transform and thinning algorithm yields a set of approximate medial axis points with associated width values. We next group spatially coherent points into tokens, as shown in Figure 6. A token is a higher order primitive which consists of 10 pixels. These tokens can then be grouped into strokes, and finally strokes from different segmentation regions may be grouped together. By operating on these higher order primitives rather than on pixels, we reduce noise and smooth the strokes.

We explored two methods for grouping tokens which differ in complexity, speed, and rendering techniques. These methods are compared in Secs 6.2 and 6.3.

6.1 Forming Tokens

The first step in tokenizing the ridge set is to classify pixels based on how many neighbors they have. Pixels are classified as follows:

orphan - no neighbors

seed - one neighbor

line - two neighbors

branch - three neighbors

Additional cases are not needed because the thinning algorithm guarantees at most 3 neighbors. During classification, queues for each type of point are created.

The tokens are grouped by performing a greedy search starting at seed points. Token objects are initialized as a single seed pixel. If the nearest neighbor to this point is also a seed, we start a new token. If the nearest neighbor point is a line point, we add the point to the token object and find the neighbor of the line point. If the nearest neighbor point is a branch point, we add the point to the token object, reclassify the point as a line point, and start a new token. Once the ridge set is grouped into tokens as in Figure 6, the tokens can be grouped into strokes.

In order to generate the color of a token, the color of each ridge point is generated by sampling the original image. A color value for the ridge point is calculated using the color ratios suggested by Schlick [20]. The color value for the token is then computed by taking a weighted average of all of the ridge points in the token.

6.2 Grouping Tokens via Moments

The first method we explored for grouping tokens into strokes was to compute the image moment of the token using the width values as weights. The first moments yield the center of mass for the token which is used as the position. The second moments allow us to construct a major and minor axis for the token which are used as the length and width of the token. We were inspired by [21].

Using this moment information, we construct a dense graph of possible merges, then reduce this graph using a variant of Prim's minimum spanning tree algorithm [5]. The graph is constructed by connecting every pair of tokens that are within a distance tolerance by an edge in the graph. A cost for each edge is computed by summing the differences of the tokens' position, orientation, and color. For example, tokens that are at right angles incur a high edge cost, while tokens that are aligned have a low edge cost. These edges are loaded into a priority queue, and one by one the lowest cost edge is removed from the queue. If this candidate merge is feasible, then the tokens are merged. A merge is feasible if the tokens haven't already been merged with other tokens. This usually means that the tokens are independent, or are at the ends of strokes. This process is repeated by until all the tokens are merged into a single stroke, or until all possible combinations of token merges have been attempted.

The merging process generates a set of strokes which cover the original set of tokens. Each token in the stroke has a width given by the minor axis of the moment of the token. The main drawback of this technique is computation time. The technique is $O(N^2)$ on the number of tokens in the image and a large amount of computation is performed for each token. However, since it is a global method it tends to produce "optimal" strokes.

6.3 Grouping Tokens via Search Cones

We also explored a second method for grouping tokens which involves a greedy search in a conical region extending from the token. For each token, search cones are created along the major axis of the token, extending out a small distance. Then the cones of every token are tested to see if they intersect the cone of any other token. Tokens with intersecting cones are merged into strokes and the process is repeated until no further merges are possible. Unmerged tokens are made into single token strokes. Next, the algorithm attempts to merge the orphan tokens into the existing strokes by testing their positions verses the search cones.

The major advantages of this method over the moment method are speed, the cone intersections can be hard coded, and merging generally takes less than $O(log(N))$ passes over the data were N is the number of tokens. In addition, in order to render this type of stroke the ridge set points can be used directly without the over head of computing B-spline curves. However, since this is a local method it may self-intersect and lack smoothness.

Figure 7: *An example of our method for drawing strokes based on moment tokens. First, points are grouped into tokens and the moment of the group is taken. Second, points are replaced by the moment centroid and additional points are added to the beginning and end of the token list. Third, the points are used as the control polygon of a B-spline curve. Forth, offset curves are computed based upon the width values. Last, the stroke is rendered.*

7 Rendering Images

Brush stroke rendering depends on the method of token grouping used in the previous phase of the algorithm. Modified versions of Strassman [24] and Pham's [17] algorithms are used to render brush strokes. Strassman and Pham modeled sumi-e brushes which taper on from a point and taper off to a point during a brush stroke. We choose to model a Filbert brush used in oil painting. Filbert brushes are good all-purpose brushes combining some of the best features of flat and round brushes [22]. To model a Filbert brush, the taper-on is constrained to a circular curve and the taper-off constrained to a parabolic curve. Examples of this type of simulated brush stroke are shown in Figure 15.

7.1 Stroke Representation

Strokes made up of tokens that are grouped using the moment method are rendered using the positions of the moments as the control points of a B-spline curve. This list of control points is called the control polygon. Since B-spline interpolation has the side effect of shortening the stroke, we add extra points on the beginning and the end to compensate (see Figure 7). In addition to the control polygon, we compute a scalar spline curve that blends the width values. Using this width spline and the control polygon, we render the strokes by drawing parallel lines in the direction of the brush stroke.

Strokes that are grouped using the search cone method are rendered using a simpler method. Normals are calculated using the finite differences of the token centers, and the widths are sampled from the nearest token. From these values a quadrilateral is generated, and filled using lines perpendicular to the normal.

This method has the advantages of speed and a great deal less computational and coding complexity than the B-spline method. However, due to the absence of blending, this method can create visible artifacts when used with alpha blending. The normal directions calculated for each of the stroke points can intersect, causing what Strassmann [24] called the "bow tie" effect. This effect is demonstrated in Figure 9. When a low alpha values are used, causing the brush stroke to appear opaque, and the brush stroke contains these self intersections, these areas of the stroke appear darker. In practice this effect is generally not noticeable with alpha values higher than 0.75.

7.2 Grouping Strokes

In addition to grouping tokens and strokes inside a single segmented region, strokes and tokens from different regions can be merged. An example is shown in Figure 8. In theory, this should smooth strokes and generate a more pleasing flow. In reality, the effect upon images varies, and may not be suitable in every case. Stroke merging can use either the moment method or the search cone method. The moment method incurs a large memory overhead due to the fact that all of the tokens and strokes from all regions need to be saved and checked. In practice this slows the system down by between one and two orders of magnitude depending on the image size and the available memory.

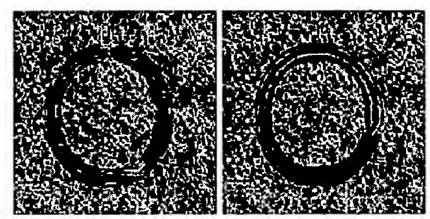


Figure 8: *A region painted without a grouping algorithm. (left image: 62 strokes), and with (right image: 1 stroke)*

8 Underpainting and Brush Effects

Underpainting is simulated by allowing the user to render strokes on top of another image. Meyer [15], Hertzmann [11], and Shiraishi et al. [21] discuss underpainting in their work. Meyer rendered brush strokes on an a background image. Hertzmann and Shiraishi blur the source image and render strokes on the blurred image.

Our system allows the user to import separate source images and underpainting images. In this way, strokes can be rendered onto a background image or onto blurred images. In addition the underpainting can be used for artistic effect as in Figure 13. The output of the system could also be used as an underpainting allowing a painting to be built up in layers.

Painting effects such as brush artifacts, paint mixing between layers, and stroke connection are also possible in the system. Alpha blending is used to simulate paints of various opacity. Paints with a high opacity will show the underlying substrate while paints with a low opacity will block the view of the substrate. The alpha parameter controls the percentage of blending between the underpainting and the brush strokes.

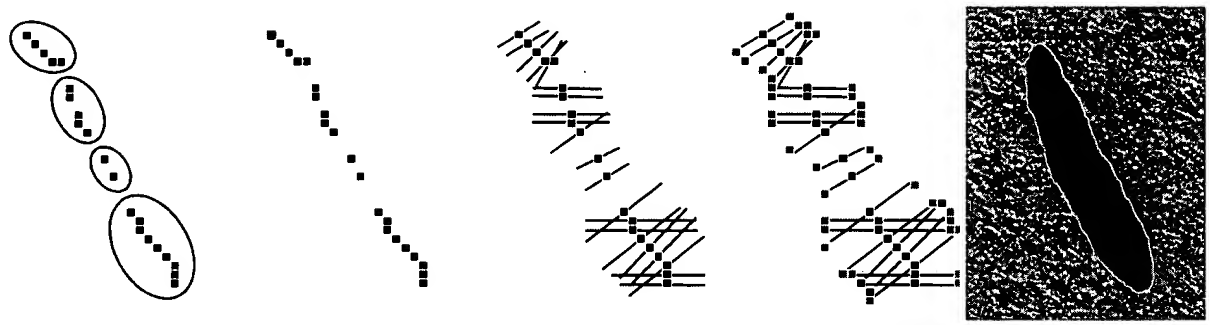


Figure 9: *An example of our method for drawing strokes based on line lists. First, points are grouped into tokens. Second, tokens are grouped into a list of points. Third, for every point a normal line is found. Forth, based on the normal directions and the width at each point edge points for the stroke are computed. Last, a brush stroke is rendered.*

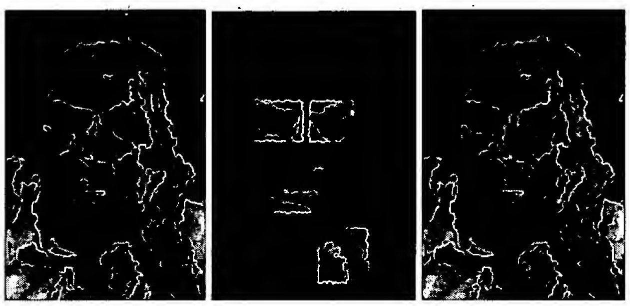


Figure 10: *An example of user-directed enhancement. The Feynman portrait is deemed by the user to lack detail in regions surrounding the eyes, mouth, and hand. The user selects these regions, and raises the segmentation level. New higher frequency strokes are drawn over the original strokes, hopefully improving the result.*

8.1 User-Directed Enhancement

Human artists often apply brush strokes in a manner which communicates the underlying three dimensional structure of the subject. Because we have no three dimensional information about the source image we implemented two heuristic techniques. The first is to start brush strokes at the widest end of the stroke. The second method allows the user to select areas of interest in the image and resegment these areas with a higher number of gray levels. As discussed in Section 4, a single set of segmentation levels is chosen for the entire image. Increasing the number of gray levels in the segmentation increases the number of strokes, which changes the impression of a painting. This difference has a strong effect on the painted image as seen in Figure 14.

The user can select areas of interest in the image and these areas are repainted with a higher number of strokes. This allows the user to direct the placement of high frequency information in the image, and in many cases improves the visual appearance of the painting. This is similar in spirit to the method of Hertzmann [10]. An example of this process is shown in Figure 10.

9 Extension: Depth Information

While working with this system, we have noticed that certain input images generated nice paintings while others do not. In particular, landscape scenes tend to work very well, while closeups (such as portraits) tend not to turn out as well, and may require manual resegmentation. We believe that there are a number of reasons for this, the primary one being painting is an inherently 3-dimensional process. Artists have spent hundreds of years refining 3D techniques, beginning in the Renaissance []. In addition, there have been numerous computer 3D painting techniques []. We applied some of these techniques to our system as a first step in incorporating 3D information into our 2D painting algorithm.

In Section 9.1, we talk about different techniques artists have developed, and in Section 9.2 we talk about how we applied these techniques to our painting system.

9.1 Artistic Techniques for Creating Space

Artists create space and distance using techniques such as perspective, detail variation, color saturation, atmospheric perspective, and warm/cool fades. The perspective effect and the detail variation are in some sense already present in images, from the mechanisms inherent in photography. We were most interested in applying the warm-to-cool fade. The warm colors - red, orange, yellow - "psychologically suggest emotion, energy and warmth while optically moving the subject to the foreground." [19] The cool colors - green, blue, violet "appear to recede." [19]

9.2 Applying these Ideas

There are a number of ways to generate depth information, including depth-from-stereo, depth-from-shading, etc. [25], as well as synthetic images, where the depth is explicitly represented in a depth buffer.

Any of these techniques will work for our system. Synthetic images have an advantage in that they are easy to obtain and have a regular, noise-free depth map. Depth-from-stereo produce somewhat noisy depth maps (based on image texture), can be hard to obtain, and often require some manual input to identify corresponding image features. However, these images are usually more visually interesting than synthetic images.

Once the depth map has been calculated, it is fed into the painting system along with the input image. The depth is used as another information channel to the segmentation process. Objects are first differentiated using the depth, and these objects are further decomposed using intensity variation. This technique was chosen because depth tends to be quite good at resolving object-object interactions, but poor at chosing how to lay strokes across a surface. Likewise, intensity can often fail to correctly identify seperate objects, but does well at placing strokes across a surface.

The depth is used to vary the levels of segmentation in the image by segmenting at a low level for distant objects and at a high level for close objects. This tends to increase the detail in close objects while decreasing detail in distant objects.

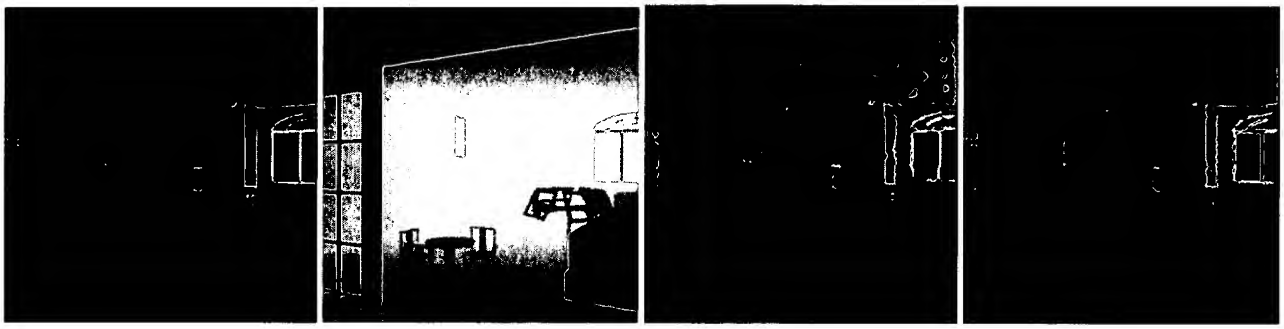


Figure 11: *An example of using depth segmentation. From left to right: the original image, the depth image, the painting without using the depth information, and the painting using depth information. No underpainting was used (for comparison)*

The algorithm then uses these segments in the same way as before, and generates a set of strokes. These strokes maintain properties such as intensity and depth. The depth is then used to modulate the color of the stroke, using the warm-to-cool fade mentioned above. In the future, we intend to also implement the color saturation and atmosperic effects. These operations are shown in Figures 11 and 16.

9.3 Depth Extension Analysis

It is clear that using depth information can increase the quality of the painting. We believe that this is because painting is an inherently 3D process, and only using image processing techniques in 2D hinders the process. However, there are several issues still to be considered.

The depth information is often noisy and incomplete, particulary if the depth map is obtained using stereo cameras or other depth-from-X techniques. This confuses the segmentation process, resulting in a noisy segmentation (and ultimately, a noisy painting). This could be handled by assigning weights to the intensity information and depth information, allowing the user (or the algorithm) to compensate for noisy or incomplete data. The technique is not applicable to portrait painting, as the depth gradient is quite small across a face. There may be other artistic techniques for portrait painting that would be more amenable to computer implementation.

We intend to explore: Are there other techniques that could be applied to computer algorithms? Are the applications of these techniques used in this system correct, or do they bias the painting unneccesarily? Are there better ways of providing tools for users to experiment? Perhaps users want more control over the process of painting, instead of less control?

10 Conclusion and Future Work

We think productive future work would include improvements to every stage of the algorithm. Images which require sophisticated segmentation, and images where viewers are sensitive to the features of the image, such as detailed portraits, can cause the method to fail. Better segmentation, such as that given by anisotropic diffusion [26], may yield immediate improvements in linking brush strokes to salient features of the image. The computation of medial axes might be less sensitive to noise if a continuous medial axis algorithm based on Voronoi partitions were used. This may also simplify the job of token-merging in our algorithm by reducing input noise. A sophisticated ordering of brush strokes, such as optimizing order based on edge correlation with the original image might improve the painting, but would come with huge cost in computation and memory usage. A more physically-based paint-mixing would give a look more reminiscent of oil painting.

Our system might also benefit from a user-assisted stage at the end to improve brush stroke ordering. An estimate of foveal attractors in the image could allow brush stroke size to be varied with probable viewer interest. Most challenging, our method could probably be extended to animated sequences, using time-continuous brush-stroke maps to ensure continuity. However, it is not clear what such animated sequences would look like, or to what extent they are useful. The most exciting future effort is to create an actual stroke-based hard copy using robotics or other technology.

11 Acknowledgments

We would like to thank Amy Gooch, Kavita Dave, Bill Martin, Mike Stark, Samir Naik, the University of Utah Graphics Group and the University of North Carolina Graphics Group for their help in this work. This work was carried out under NSF grants NSF/STC for computer graphics EIA 8920219, NSF/ACR, NSF/MRI and by the DOE AVTC/VIEWS.

Appendix: Thinning Algorithm

Rosenfeld's thinning algorithm tests each pixel neighborhood for redundant pixels. Since this test is performed at each pixel in the image, it is important that it be fast. We have developed an original technique for determining whether a pixel is redundant.

In this Appendix, we describe the test that is performed at each pixel. The input is the pixel's 8-neighborhood, and the output is a boolean telling whether this pixel is redundant. The idea of this test is to construct a graph of all of the paths through this neighborhood, then test whether removing the center pixel breaks any of these paths. If it does not, then the center pixel represents a redundant path. If it does, then the center pixel must remain (this is called *8-simple*).

Observations

The first step is to construct a graph $G(V, E)$ that connects every pixel, except for the center pixel, to its adjacent pixels. This is illustrated in Figure 12.

This graph G represents the paths through this neighborhood. The test for 8-simpleness just becomes a test for the connectedness of G. We can use Euler's Theorem for connected planar graphs, which states that $v + r - 2 = e$, where v denotes vertices, r denotes regions of plane, e denotes edges. Rearranging the terms yields two conditions for 8-simpleness in a pixel neighborhood: 1) there can be no isolated pixels; 2) $v - 1 \leq e$.

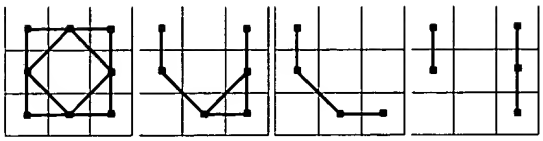


Figure 12: *From left to right: The graph representation of a full neighborhood. An 8-simple neighborhood. Another 8-simple neighborhood. A non-8-simple neighborhood.*

Implementation

Let the edges of G be E_i, and a graph representation of a full neighborhood be N. Then the algorithm is:

```
for each i ∈ G {
  //Test for isolated pixels
  if ( E_i ∩ N = ϕ ) return not_simple;
  // Else add to the total edges
  else edges += degree( E_i ∩ N ) ;
}
// Euler's Thereom.
if ( edges ≥ v − 1 ) return simple;
else return not_simple
```

Notes

We encoded the E_i and N in binary, resulting in a fast test. The only storage requirements are the eight sets E_i, which can be stored as eight integers. Most previous thinning algorithms in the computer graphics literature enumerate and store every case, resulting in a large overhead.

References

[1] BERMAN, D. F., BARTELL, J. T., AND SALESIN, D. H. Multiresolution painting and compositing. *Proceedings of SIGGRAPH 94* (1994), 85–90.

[2] BLUM, H. A transformation for extracting new descriptions of shape. *Models for the Perception of Speech and Visual Form* (1967), 362–380.

[3] BURBECK, C. A., AND PIZER, S. M. Object representation by cores: Identifying and representing primitive spatial regions. *Vision Research 35*, 13 (1995), 1917–1930.

[4] COCKSHOTT, T. *Wet and Sticky: A Novel Model for Computer-Based Painting*. PhD thesis, University of Glasgow, 1991.

[5] CORMEN, T., LEISERSON, C., AND RIVEST, R. *Introduction to Algorithms*. MIT Press, 1990.

[6] CURTIS, C. J., ANDERSON, S. E., SEIMS, J. E., FLEISCHER, K. W., AND SALESIN, D. H. Computer-generated watercolor. *Proceedings of SIGGRAPH 97* (August 1997), pages 421–430.

[7] GOOCH, B., SLOAN, P.-P. J., GOOCH, A., SHIRLEY, P., AND RIESENFELD, R. Interactive technical illustration. *1999 ACM Symposium on Interactive 3D Graphics* (April 1999), 31–38.

[8] HAEBERLI, P. E. Paint by numbers: Abstract image representations. *Proceedings of SIGGRAPH 90 24*, 4 (August 1990), 207–214.

[9] HEARN, D., AND BAKER, M. P. *Computer Graphics*. Prentice-Hall, 1986.

[10] HERTZMANN, A. Paint by relaxation. In *Computer Graphics International 2001* (July 2001), pp. 47–54. ISBN 0-7695-1007-8.

[11] HERTZMANN, A. Painterly rendering with curved brush strokes of multiple sizes. *Proceedings of SIGGRAPH 98* (July 1998), 453–460.

[12] HUBBARD, P. M. Approximating polyhedra with spheres for time-critical collision detection. *ACM Transactions on Graphics 15*, 3 (July 1996), 179–210.

[13] JAIN, R., KASTURI, R., AND SCHUNCK, B. *Machine Vision*. McGraw-Hill, 1995.

[14] LITWINOWICZ, P. Processing images and video for an impressionist effect. *Proceedings of SIGGRAPH 97* (August 1997), 407–414.

[15] MEIER, B. J. Painterly rendering for animation. *Proceedings of SIGGRAPH 96* (August 1996), 477–484.

[16] PERLIN, K., AND VELHO, L. Live paint: Painting with procedural multiscale textures. *Proceedings of SIGGRAPH 95* (August 1995), 153–160.

[17] PHAM, B. Expressive brush strokes. *Computer Vision, Graphics, and Image Processing. Graphical Models and Image Processing 53*, 1 (Jan. 1991), 1–6.

[18] ROSENFELD, A. A characterization of parallel thinning algorithms. *InfoControl 29* (November 1975), 286–291.

[19] SASAKI, H. Color psychology. *http://www.shibuya.com/garden/colorpsycho.html* (April 1991).

[20] SCHLICK, C. Quantization techniques for visualization of high dynamic range pictures. *Proceedings of the 5th Eurographics Workshop* (June 1994), 7–20.

[21] SHIRAISHI, M., AND YAMAGUCHI, Y. An algorithm for automatic painterly rendering based on local source image approximation. *NPAR 2000 : First International Symposium on Non Photorealistic Animation and Rendering* (June 2000), 53–58.

[22] SMITH, A. R. Varieties of digital painting. Tech. rep., Microsoft Research, August 1995.

[23] SOUSA, M. C., AND BUCHANAN, J. W. Computer-generated graphite pencil rendering of 3d polygonal models. *Computer Graphics Forum 18*, 3 (September 1999), 195–208.

[24] STRASSMANN, S. Hairy brushes. *Siggraph 20*, 4 (Aug. 1986), 225–232.

[25] TRUCCO, E., AND VERRI, A. *Introductory Techniques for 3-D Computer Vision*. Prentice-Hall, 1998.

[26] TUMBLIN, J., AND TURK, G. LCIS: a boundary hierarchy for detail-preserving contrast reduction. *Proceedings of SIGGRAPH 99* (August 1999), 83–90. ISBN 0-20148-560-5. Held in Los Angeles, California.

[27] WILLIAMS, L. 3D paint. *1990 Symposium on Interactive 3D Graphics* (1990), 225–233.

[28] WONG, E. Artistic rendering of portrait photographs. Master's thesis, Cornell University, 1999.

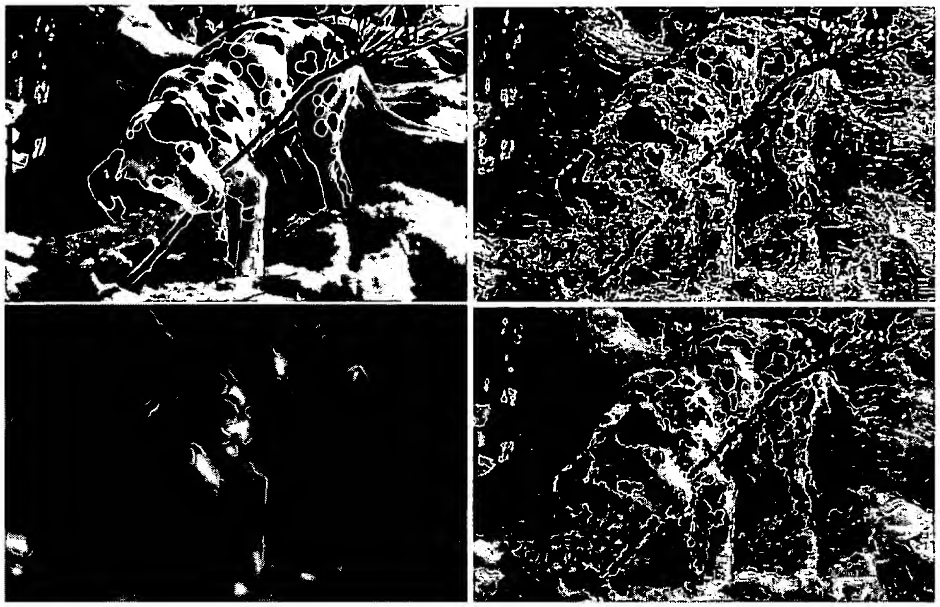


Figure 13: *Underpainting is a method used by artists to block in basic forms and values in a painting. We simulate underpainting by allowing the user to render strokes on top of another image. This example shows a source image. Next a painting made from this image using the source image as an underpainting. The third image is an underpainting made by changing the color gamut of the original image and then blurring the image. The final painting was made by painting strokes, using the first image as a source, onto the modified underpainting. This technique can be expanded to create images with multiple painted layers.*

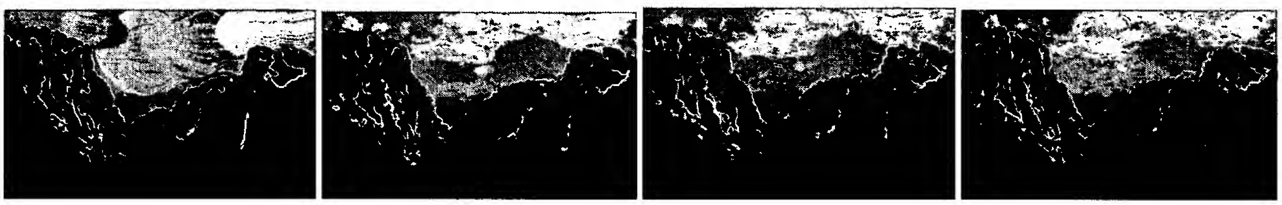


Figure 14: *An example of varying the number of gray levels in the segmentation and the resulting paintings.From top to bottom the images were segmented using; 12, 48, 72, and 150 gray levels.*

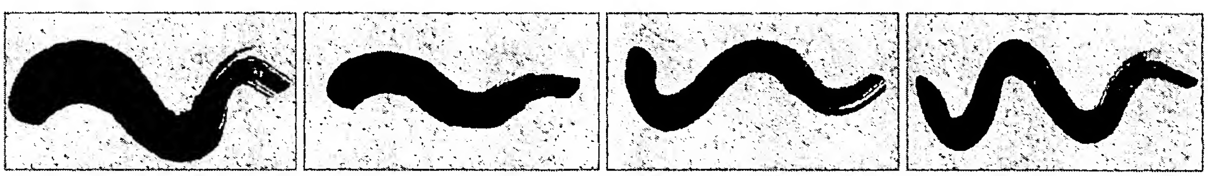


Figure 15: *Digitally simulated brush strokes. These images demonstrate the range of brush effects.*

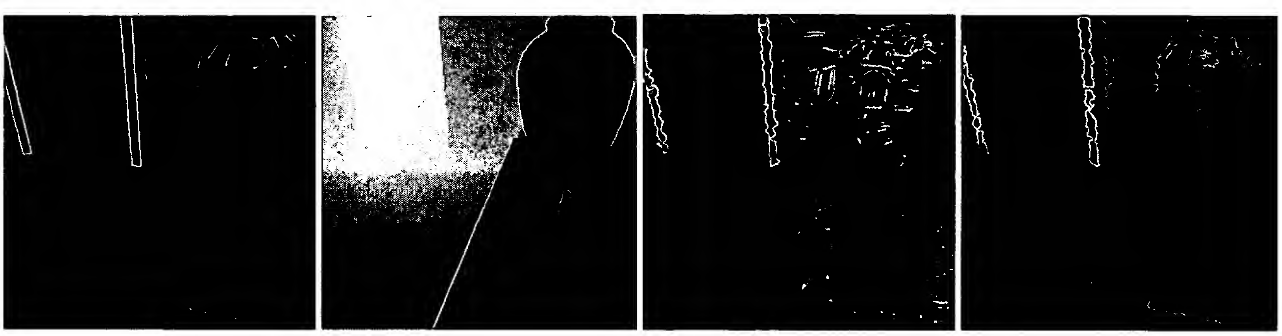


Figure 16: *An example of using depth segmentation. From left to right: the original image, the depth image, the painting without using the depth information, and the painting using depth information. No underpainting was used (for comparison)*

Animating Images with Drawings

Peter Litwinowicz†
Lance Williams†
Apple Computer, Inc.

ABSTRACT

The work described here extends the power of 2D animation with a form of texture mapping conveniently controlled by line drawings. By tracing points, line segments, spline curves, or filled regions on an image, the animator defines features which can be used to animate the image. Animations of the control features deform the image smoothly. This development is in the tradition of "skeleton"-based animation, and "feature"-based image metamorphosis. By employing numerics developed in the computer vision community for rapid visual surface estimation, several important advantages are realized. Skeletons are generalized to include curved "bones," the interpolating surface is better behaved, the expense of computing the animation is decoupled from the number of features in the drawing, and arbitrary holes or cuts in the interpolated surface can be accommodated. The same general scattered data interpolation technique is applied to the problem of mapping animation from one image and set of features to another, generalizing the prescriptive power of animated sequences and encouraging reuse of animated motion.

Keywords: Image warping, animation, scattered data interpolation.

Background

Rich detail and texture are usually reserved for the background paintings in an animation. Production economics do not permit the foreground figures to be dressed in plaid, for example, and such effects are difficult to achieve by traditional means. Three-dimensional computer animation offers shading and texture, but the stylization of form possible in traditional cel animation has proved more elusive. Much of the motivation for the work described here comes from the technique of traditional animation, where all action is portrayed by drawings -- points, lines, and curves -- defined at arbitrary instants of time. Interpolation defines the full sequence from the sparse keys. We envision a similar interpolation in space, embedding the objects and characters the artist has drawn in a surface controlled by the lines of the drawings. The interpolated motion of corresponding keyframe drawings is used to define spatial deformations which may be applied to other images. "Feature-based" deformations, controlled by the motion of arbitrarily-placed points, curves, and regions, offer a direct and natural method of animating complex images and forms. An earlier attempt at animating drawings by their features [Litw91] required the user to define a mesh of bilinear Coons patches [Forr72]. The curved boundaries of the patches could be aligned with features of interest to the animator, and subsequently animated to control the image. Although the Coons patches are inexpensive to evaluate, the manual division of the image into a mesh, and the necessity of animating all of the patch boundaries to control the motion, require substantial time and effort. Specifying and animating only the features of interest is both vastly more general and a great deal easier for the animator.

Deformations based on tensor-product splines [Sed86][Farin90] are actually a more recent development than "feature based" deformations defined by line segments, which were introduced by Burtnyk and Wein in [Burtn76]. The goal of that work was to permit an animated "skeleton" of linked line segments to drive the animation of a drawing, in this case by polygonal tessellation of regions around the "bones" of the skeleton. An alternate parametrization, based on a skeleton derived from the shape of the matte or support of the image region to be animated, has been described by Wolberg [Wolb89]. In this case, the skeleton is the result of successive thinning operations applied to the original shape. The image warping algorithm is specialized for morphing, that is, for transforming between two image/shapes. Driving an image warp by modifying the skeleton alone would require a slightly different algorithm. Automatic "medial axis" skeletons of this type might be useful for some purposes, but there is no guarantee that the "bones" will align with features the animator is interested in controlling directly.

More recent skeleton animation work appeals to smoother interpolation functions. Van Overveld [vanO90] describes a physical simulation which is calculated for a simple skeleton, then applied to a more complex model by a distance-weighted "force field." The field is defined by a dense set of points on the limbs of the skeleton. The formula used is equivalent to Shepard's interpolation, a simple scattered data interpolant originally developed for terrain surfaces [Shep68]. In [Beier92], Beier and Neely developed an algorithm for image morphing based on Shepard's interpolation, with a significant novelty: the control primitives were extended to include line segments as well as points. Since the line segments can be aligned with important edges in the image, the metamorphosis was termed, "feature based." At edge-like features of the image, a single line segment does the work of dozens of points, and offers a natural and intuitive means of interpolating local orientations.

Thin-plate spline surfaces were introduced to computer-aided geometric design by Harder and Desmarais [Hard72]. Application of finite-element methods to computing smooth surfaces over

†c/o Apple Computer, Inc.
1 Infinite Loop, MS 301-3J
Cupertino, CA 95014 USA
email: Pete Litwinowicz: litwinow@apple.com
Lance Williams: lance.w@apple.com

scattered data for CAGD purposes was first essayed by Pilcher [Pilch74]. Smooth scattered data interpolants, introduced as analogues of physical surfaces, have more recently been applied in vision and image reconstruction [Grim81]. The demands of rapid processing for practical vision systems has motivated attempts to compute some of these surfaces using fast numerical methods [Terz88], and our animation system utilizes these techniques. We have implemented a system which performs the scattered data interpolation for animated deformations or morphing by using multigrid finite-difference evaluation of a thin-plate spline surface. This approach extends the "feature" primitives to curves and solid regions, realized as densely sampled points. In addition to generalizing the control primitives, the underlying surface which defines the deformation is better behaved than a Shepard's interpolant. This is particularly important when deformations are used for animation, without the texture interpolation invoked in a "morph."

Description of the problem and our solution

Given starting and ending shapes for a set of primitives in the plane, such as curves, lines and points, we would like to calculate a warp that transforms regions between the primitives in a well-behaved and intuitive way. By aligning curves, lines and points with features in an image, intuitive controls for image warping are easily constructed. Deformation of an image can then be accomplished by applying the warp defined by the original drawing and any other drawing of the same features. It is then possible to animate an image simply by animating the drawing, and applying the corresponding image warp at each frame.

From a control primitive's original and final shape we can derive a set of displacements. For a point the displacement is simply a $(\Delta x, \Delta y)$ pair. For a line or polyline, continuous displacements all along the length are defined by the initial and final shape. Not only the "skeleton animation" of Burtnyk and Wein [Burtn76], but a number of subsequent facial animation systems and morphing programs are based on triangulation of displaced points [Gosh86]. For interpolating a set of scattered points, Delaunay triangulation is frequently used. A triangulation is defined for the original feature set, then the vertices are interpolated toward the final shape, and the triangles texture-mapped from the original image. While rapid to compute, the warp is generally not as smooth as desired. The triangulation can be seen in the resulting animation, as the texture map shears along the edges of the triangles.

Beier and Neely [Beier92] advanced a modified Shepard's interpolant which added line segments as control primitives. This method interpolates displacements using a distance-weighted technique and produces smoother interpolations than triangulation. The usual difficulty with these distance-weighted interpolants is trading off "cusps" against "flats" at the data points. In addition, the interpolation may become very expensive as the number of primitives increases, since each contributes at every point on the surface. To give the animator local control, Beier and Neely associate a finite region of influence -- a threshold distance from a point or line segment -- with each primitive. The process of specifying the region size for each primitive can potentially be tedious, and for many warps, no combination of region extents and inverse-distance weighting exponents yields the desired result [Rupr92].

Instead of explicit control over the size of the basis functions used in the interpolation process, our goal was to provide a technique which automatically extended regions of influence to the next user-defined primitive. Another goal was to have a nice "smooth" interpolant, but at the same time provide a mechanism for intuitively introducing discontinuities in the interpolant where appropriate. Finally, we wished to provide curves as deformation primitives.

The thin-plate spline provided a nice compromise for our goals. The region of influence for a particular primitive is global, but the region most affected is the area between a primitive and its nearest neighbors. The thin-plate spline is C^1 continuous, certainly smoother than a piecewise planar triangulated surface, and not so potentially cuspy as a Shepard's interpolant.

The thin-plate spline is one solution to a class of scattered data interpolation problems that have the following problem statement (from [Franke79]): "Construct a smooth bivariate function, $F(x,y)$, which takes on certain prescribed values, $F(x_k,y_k) = f_k$, $k = 1, \ldots, N$. The points (x_k,y_k) are not assumed to satisfy any particular conditions as to spacing or density, hence the term 'scattered.'"

How does our problem map onto the scattered data problem? For a number of known (x_k,y_k) positions in the image plane, we have known displacements $(\Delta x_k, \Delta y_k)$ as defined by our original and destination drawings. Substituting Δx_k for f_k above, we calculate a smooth interpolating function for the x-displacements for an entire image, and similarly for the y-displacements. The thin plate also has the added constraint that the surface everywhere should minimize the following smoothness functional:

$$\iint_{\Omega} \frac{\partial^2 F}{\partial^2 u} + 2\frac{\partial^2 F}{\partial u \partial u} + \frac{\partial^2 F}{\partial^2 v} \, du dv$$

where Ω is the domain of the surface, and F is the surface itself. Encoded in Ω are the cuts and holes in the surface.

The thin plate spline can be solved by using a $d^2 \log d$ basis function at each point (where d is the distance from the point), and solving the linear system. This becomes extremely expensive as the number of known points increases. By solving the problem on a discrete grid, the solution time is dependent on the strain energy in the plate and not on the number of data points (beyond a small initialization cost). Another advantage to discretizing the problem is that discontinuities in the interpolant are easy to handle. In the continuous problem, it is not obvious how to change the basis function $d^2 \log d$ to handle irregular discontinuities. Our grid sizes are on the order of the image size, in pixels; we make sure that we have at least one grid element per pixel in the image. Finally, we use a coarse-to-fine multiresolution method to calculate our interpolants [Terz88].

We present the animator with curves, polylines, and points as deformation primitives. When solving the problem on a discrete grid we must scan convert the primitives' displacements onto the grid. In practice, we discretize the primitives into equidistant samples.

The animator specifies discontinuities in the surface by supplying an extra black-and-white matte; an image pixel is "connected" to neighboring pixels labeled "nonzero" in this matte. For most purposes, the animator uses the ordinary alpha matte in the role of discontinuity matte as well, but they may be specified separately. For instance, the eyes and mouth of the characters in Figure (1) are on a separate cel level, with holes specified in the top layer. These holes also specify discontinuities in the interpolant, so when the top eyelid closes, it does not affect the lower eyelid.

In applying the displacements we use a forward mapping technique, as opposed to the inverse mapping technique implemented by Beier and Neely (the former is a "many to one" mapping, the latter, "one to many"). All the warped images in the color plates illustrating this paper were generated using a forward mapping, including the pictures demonstrating Beier-Neely interpolation. The Beier interpolation picture actually uses 2 points for the interior points and four lines along the edges; the displacements for the points are weighted with Shepard's formula and the edges with the Beier-Neely modifications for lines. The checkerboards warped with thin-plate interpolation have displaced interior feature points and four stationary lines along the edges.

The interpolated displacements for the entire surface are applied to the image at each pixel. The image is rendered as a polygon mesh; each original pixel becomes a polygon vertex in the mesh, except where the discontinuity matte breaks the connections. Subpixel positioning of the displaced quadrilateral endpoints is important for good results.

Observations

Our experience suggests that the imposition of a structure to animate the image, such as a grid or mesh of polygons, can impose a heavy burden on the animator. It is far more intuitive to specify, and animate, a simple drawing which parametrizes and controls the image.

Polygonal texturing may not be smooth enough for the extreme deformations used in animation. Distance-weighted interpolants may not be smooth enough, either, and may limit the number of control primitives for practical purposes.

A very valuable feature of the thin-plate spline surface is its *idempotency*. New features can be added at any time without modifying the current mapping, and subsequently serve as handles for further animation. With a polygonal mapping, this can be ensured by subdividing only triangles in which new control points are introduced. With a distance weighted mapping, this property is impossible to achieve, and one must settle for gradually, over time, blending in the contribution of newly introduced control features.

The iterative relaxation used to compute the multigrid spline surface can profit from frame-to-frame coherence in animation. By using the last surface computed as an estimate for the next frame, the expense for the sequence is greatly reduced. The first frame of the example animated cat sequence took 5 min., 34 sec.; subsequent frames, on average, 3 min., 30 sec. (surface computed as a 513x513 grid on a MIPS 36Mhz R3000).

There are several ways to trade off computation and quality in the surface. One is to evaluate the surface on a coarse grid, and use tensor-product interpolation to upsample it. Another is to increase the error permitted when iteration is ceased, or to perform a fixed number of iterations. In this case, a modified form of Southwell iteration [Gera94] offers improved results for the same number of cycles. We implemented this option at the suggestion of Eric Chen, who was inspired by the "shooting" method of computing radiosity [Gortler93].

Acknowledgments

The authors would like to extend heartfelt thanks to Laurence Arcadias for the cartoon face animated in our illustrations, and to Subhana Ansari for layout.

References

[Beier92] Beier, T. and S. Neely. "Feature-Based Image Metamorphosis," Computer Graphics, Volume 26, Number 2, July 1992, pp. 35-42.

[Burtn76] Burtnyk, N. and M Wein. "Interactive Skeleton Techniques for Enhancing Motion Dynamics in Key Frame Animation," CACM, Vol. 19, Number 10, October 1976.

[Farin90] Farin, G. Curves and Surfaces for Computer Aided Geometric Design, A Practical Guide. Second Edition. Academic Press, NY, 1990.

[Forr72] Forrest, A. "On Coons and Other Methods for the Representation of Curved Surfaces," Computer Graphics and Image Processing, 1, 1972, pp. 341-369.

[Franke79] Franke, R. "A Critical Comparison of Some Methods for Interpolation of Scattered Data," Report NPS-53-79-03 of the Naval Postgraduate School, Monterey, CA. Obtained from U.S. Department of Commerce, National Technical Information Service.

[Gera94] Gerald, Curtis and P. Wheatley, Applied Numerical Analysis, Fifth Edition, Addison-Wesley Publishing Company, pp. 159-164.

[Gortler93] Gortler, Steven, Michael Cohen and Philipp Slusallek, "Radiosity and Relaxation Methods: Progressive Refinement is Southwell Relaxation," Princeton University Research Report CS-TR-408-93, February 1993.

[Gosh86] Goshtasby, A., "Piecewise Linear Mapping Functions for Image Registration," Pattern Recognition 19:6, 1986, pp. 459-466.

[Grim81] Grimson, W., From Images to Surfaces, MIT Press, 1981.

[Hard72] Harder, R., and R. Desmarais, "Interpolation Using Surface Splines," J. Aircraft, Vol. 9, February 1972, pp. 189-191.

[Litw91] Litwinowicz, P. "Inkwell: A $2^1/_2$-D Animation System," Computer Graphics, Volume 25, Number 4 , 1991, pp. 113-121.

[Patterson91] Patterson, E., P. Litwinowicz and N. Greene, "Facial Animation by Spatial Mapping," Computer Animation 1991, Springer-Verlag, NY 1991, pp. 31-44.

[Pilch74] Pilcher, David, "Smooth Parametric Surfaces," in Computer Aided Geometric Design, by Barnhill and Riesenfeld, Academic Press, NY, 1974, pp. 237-253.

[Rupr92] Ruprecht, D., and H. Müller, "Image Warping with Scattered Data Interpolation Methods," Research Report 443, Dortmund University, November 6, 1992.

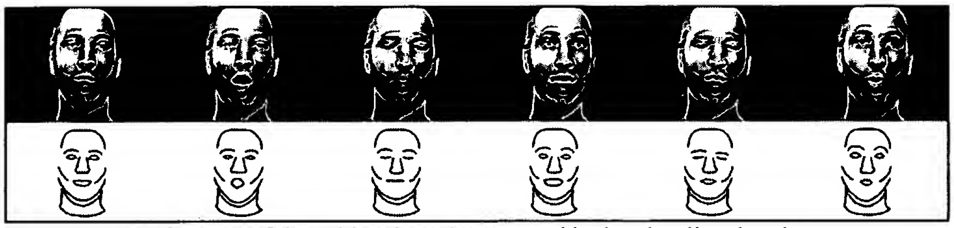
[Shep68] Shepard, D., "A Two-Dimensional Interpolation Function for Irregularly Spaced Data," Proceedings of the 23rd Nat. Conf. ACM, 1968, pp. 517-523.

[Sed86] Sederberg, T. and S. Parry. "Free-Form Deformation of Solid Geometric Models," *Computer Graphics,* Volume 20, Number 4, August 1986, pp. 151-160.

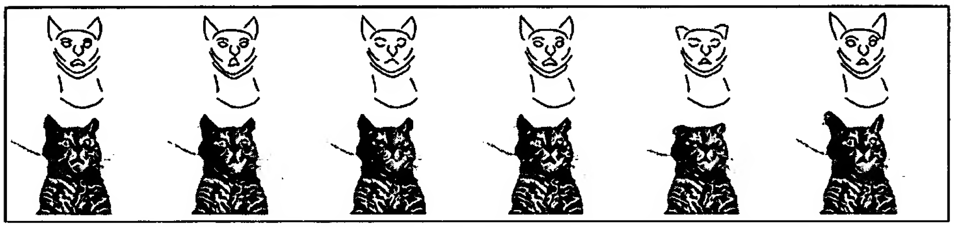
[Terz88] Terzopoulos, D. "The Computation of Visible-Surface Representations," IEEE Transactions on Pattern Analysis and Machine Intelligence, Vol. 10, No.4, July 1988.

[Wolb89] Wolberg, George. "Skeleton Based Image Warping," The Visual Computer, Vol. 5, Number 1/2, March 1989, pp. 95-108.

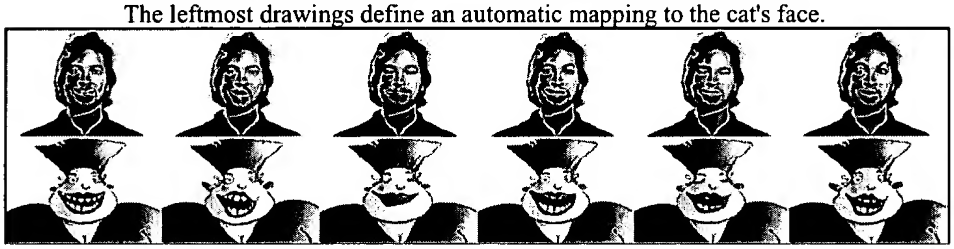
[vanO90] van Overveld, C. W. A. M., "A technique for motion specification in computer animation," The Visual Computer, Number 6, 1990, pp. 106-116.



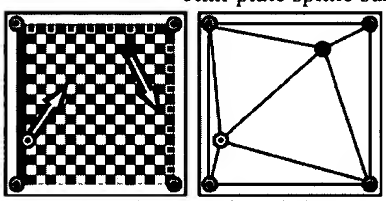
Key features of the subject have been traced by hand as line drawings.



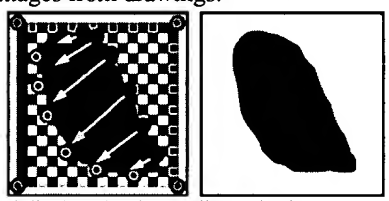
The leftmost drawings define an automatic mapping to the cat's face.



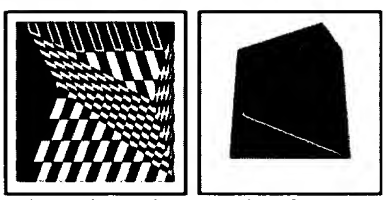
Thin-plate spline surfaces animate images from drawings.



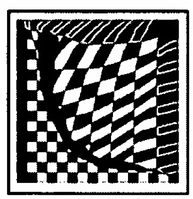
2 displaced points; triangulation.



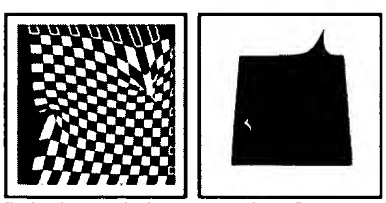
6 displaced points; discontinuity map.



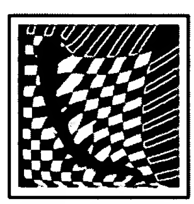
Polygonal mapping; ΔX surface.



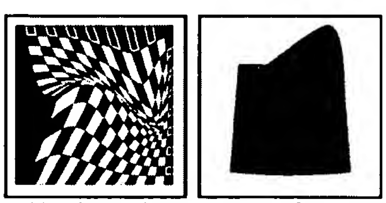
Polygonal interpolation.



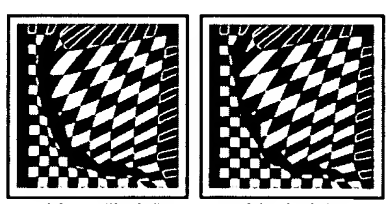
Beier interpolation; ΔX surface.



Beier interpolation.



Thin-plate spline; ΔX surface.



without "hole" with "hole"
Thin-plate spline.

Animation

The top row of pictures shows sample frames from a video sequence. Key features of the subject have been rotoscoped to extract animated line drawings. Using cross-synthesis procedures described in [Patterson91], and the line drawings of the actor and cat in the leftmost column, the animation for the other line drawings has been automatically generated for the cat. Using these animated line drawings and the original cat photograph (shown in the leftmost column), thin-plate spline surfaces are used to compute each frame of the animation. The final two rows of frames, at left, show two more characters animated in this way, with the original faces shown in the leftmost column. The eyes and teeth are animated on a separate layer.

Holes and Cuts

The two columns of pictures on the far left show the relative smoothness of image warps based on various scattered data interpolation methods. At near left, we show the process of introducing deliberate "holes" or "cuts" in interpolating surfaces:

Top left test image (closing gap in checkerboard). The six red control points are moved to the tips of the yellow arrows. The blue points, as well as each edge of the square, are held in place. Top right, the map which controls the continuity of the surface.

Polygonal interpolation. Triangulation results in sharp bends within the checkerboard as it stretches. Interpolation is local, however, with no influence across the gap.

Beier interpolation. Folds and creases appear in the checkerboard, and the left edge of the hole folds under as the right edge approaches. The "hole" in this and the next example is simply a matte (transparency map) which makes part of the surface transparent.

Thin-plate spline, with no holes in the surface. As the right edge is stretched, the left edge folds under.

Thin-plate spline, with an actual "hole" in the interpolated displacement surface. As the right edge is stretched, the left edge is relatively unaffected. This type of control is necessary for the animation examples at the top of the page. When closing an eyelid, the animator doesn't want to affect the region below the eye. Continuity and translucency are specified with independent maps.